Calculator with a GUI

Emin Musayev

CS 413-02 Spring 2020

Project Repository: https://github.com/csc413-02-FA2020/csc413-p1-emusayev

Introduction

The purpose of this assignment in this Software Development class at SFSU is to help me computer arithmetic expression in an GUI calculator application using Java. In this case, we had 4 main class file that consisted most of the program we had to work with which is Operand.java, Operator.java with operator subclasses, Evaluator.java where we had to write stack algorithm to compute expression, but also to test these programs on the console, using unit test on Evaluation Driver and implement the correct GUI in EvaluationUI to not only run these application successfully , yet also be able to compute and calculate expression such as addition , multiplication and other operator that were required in the assignment with the correct priority to be able to produce the right output.

Instructions

To run this application, you will need the require the most recent up to date IntelliJ Community 2020.02 along with the corresponding Java SE 11.04. Once all of these is set up to computer, we will need to clone the repository and import it to IntelliJ IDE. From there it is a matter of right clicking any of operator and evaluator to run you program which is mainly Evaluator UI and Evaluation Driver. Even though this is not the best way to open a calculator on computer, these projects were a great review to help me use and implement HashMap and Stack

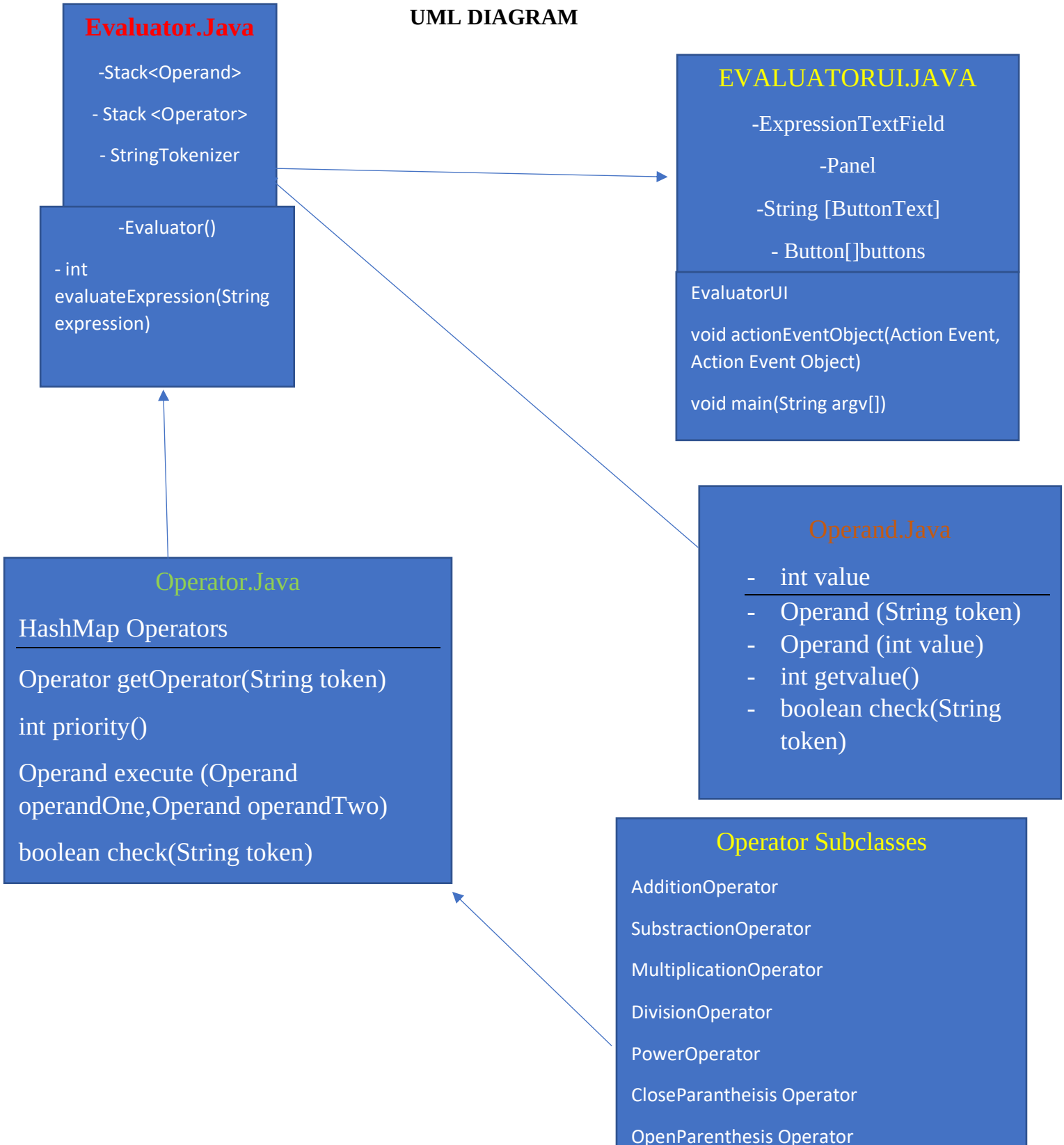
Java SE JDK 11.04

https://www.oracle.com/downloads/licenses/javase-license1.html

Implementation

As we previously mention in the intro, the main part of this application is in four class which can be visualized in UML Diagram. The main idea of this program that between two stacks, one containing Operator object and another containing Operand object you are using these two objects. In fact, I realized earlier in the program that a string expression must be scanned either by unit test which we were given or inputted manually through the console (which needs to be implemented). As a result, the string expression will break through StringTokenizer breaking into single character tokens and will call Operator and Operand class to determine and check whether the type of char that token represented in the expression. After the token is scanned and the type was finally determined by Evaluator.Java it will run the token against series of rules that were predetermined already mainly from the type of token and the content of stacks its getting information to evaluate expressions successfully.

UML DIAGRAM


Evaluator.Java
-Stack<Operand>
- Stack <Operator>
- StringTokenizer
-Evaluator()
- int evaluateExpression(String expression)
EVALUATORUI.JAVA
-ExpressionTextField
-Panel
-String [ButtonText]
- Button[]buttons
EvaluatorUI
void actionEventObject(Action Event, Action Event Object)
void main(String argv[])
Operand.Java
- int value
- Operand (String token)
- Operand (int value)
- int getvalue()
- boolean check(String token)
Operator.Java
HashMap Operators
Operator getOperator(String token)
int priority()
Operand execute (Operand operandOne,Operand operandTwo)
boolean check(String token)
Operator Subclasses
AdditionOperator
SubstractionOperator
MultiplicationOperator
DivisionOperator
PowerOperator
CloseParantheisis Operator
OpenParenthesis Operator

Operator

HashMap OPERATORS

HashMap instance are known to be inside abstract class Operator. Thus, the private static HashMap contains token as a key and Operator subclasses as values. All Operator have their own subclasses and it will be called within this HashMap.

Operator getOperator(String token)

- This is an accessor method for HashMap Operator return Operator equivalent to a token.

int priority()

- This is an abstract class defined in Operator Subclasses, to return priority values

Operand execute(Operand operandOne, Operand operandTwo)

- Defined in Operated Subclasses it takes two number otherwise in our case of program Operands and basically does mathematical arithmetic computation solely based on the current Operator

boolean check(String token)

- If a parsed token is valid Operator symbol like * or / return true if token is an operator that was specified in HashMap otherwise return false.

Operator Subclasses

- We did have all operator in just one single class we had all inherit from the Operator, represented in Operator that can take such as these ones +,-,/,*,^ and more .

Operand

int value

Private date field representing an int value of Operand.

int getValue()

Accessor method for value and pretty much return value.

boolean check(String token)

This method checks whether a token is valid operand.

Evaluator

Stack<Operand>

Stack<Operator>

- Evaluator contained two private stacks, and both contained objects since their sole purpose is to break this up into String expression into a form that can be broken down into individual operations.

StringTokenizer

- String Tokenizer breaks down expression into single character Strings to be able to parse one at a time.

String DELIMETERS

- Constant contains character to recognize for parsing.

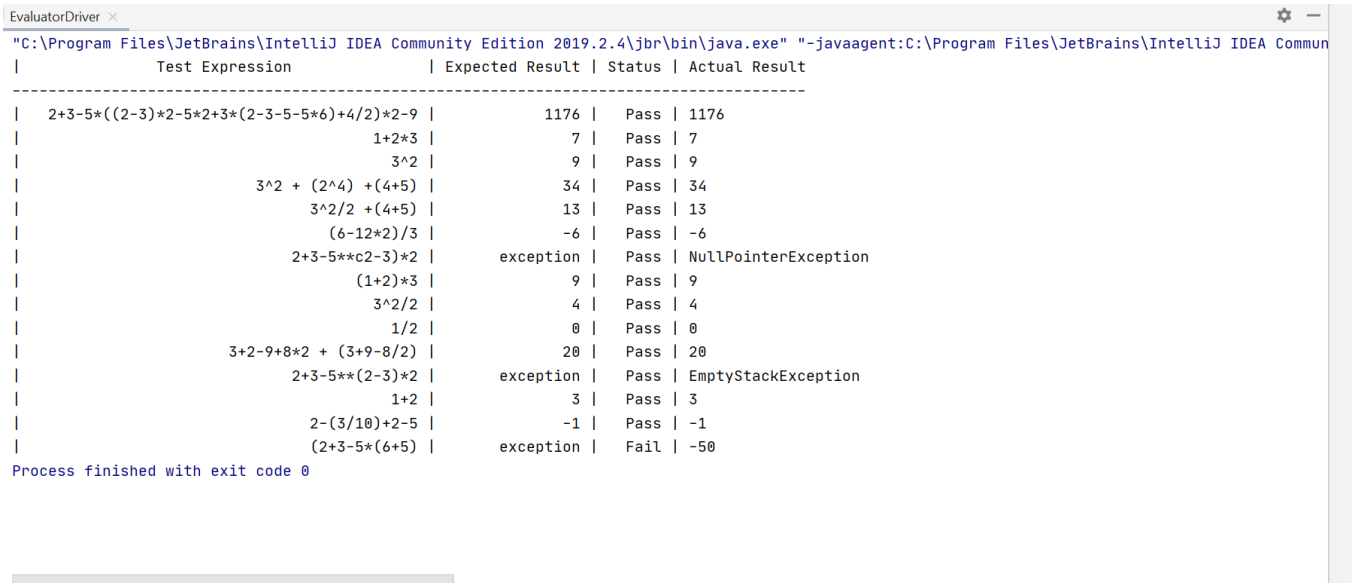
int evaluateExpression(String expression)

- This is the method where most of our stack algorithm resides it is well commented and explained .

Results/Conclusion

When originally working around the preexisting skeleton code in Evaluator class it threw me off for a while when building this program if you look at our class Slack Channel there was a point in which I was getting NullPointerExpection error and from there I decided to rewrote the whole program from beginning . I found that to be the easiest for me to figure out my problem as it helped me out understand the program much more clearly instead of dwelling on my old previous clustered code. In the end, I found my problem was my misunderstanding how the stack functions between operandOne and operandTwo confusing them in their orders when pushing and pop. After debugging for hours, I find the answer to be very simple and easy on which one should comes first when dealing with complex expressions in the EvaluationDriver Test. As a result, I managed to write a program that passes all conditions with no failure as shown in the picture below.

As of this documentation there aren't any known bug that I have encountered in the testing after it has been thoroughly looked at the tester, compiler, and the GUI.


EvaluatorDriver
"C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019.2.4\jbr\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Commun
| Test Expression | Expected Result | Status | Actual Result
| 2+3-5*((2-3)*2-5*2+3*(2-3-5-5*6)+4/2)*2-9 | 1176 | Pass | 1176
| 1+2*3 | 7 | Pass | 7
| 3^2 | 9 | Pass | 9
| 3^2 + (2^4) +(4+5) | 34 | Pass | 34
| 3^2/2 +(4+5) | 13 | Pass | 13
| (6-12*2)/3 | -6 | Pass | -6
| 2+3-5**c2-3)*2 | exception | Pass | NullPointerException
| (1+2)*3 | 9 | Pass | 9
| 3^2/2 | 4 | Pass | 4
| 1/2 | 0 | Pass | 0
| 3+2-9+8*2 + (3+9-8/2) | 20 | Pass | 20
| 2+3-5**(2-3)*2 | exception | Pass | EmptyStackException
| 1+2 | 3 | Pass | 3
| 2-(3/10)+2-5 | -1 | Pass | -1
| (2+3-5*(6+5) | exception | Fail | -50
Process finished with exit code 0